



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Eman Babar

Reg. number:

23-NTU-CS-FL-1148

Semester:

5th- A

Assignment 1: Operating System

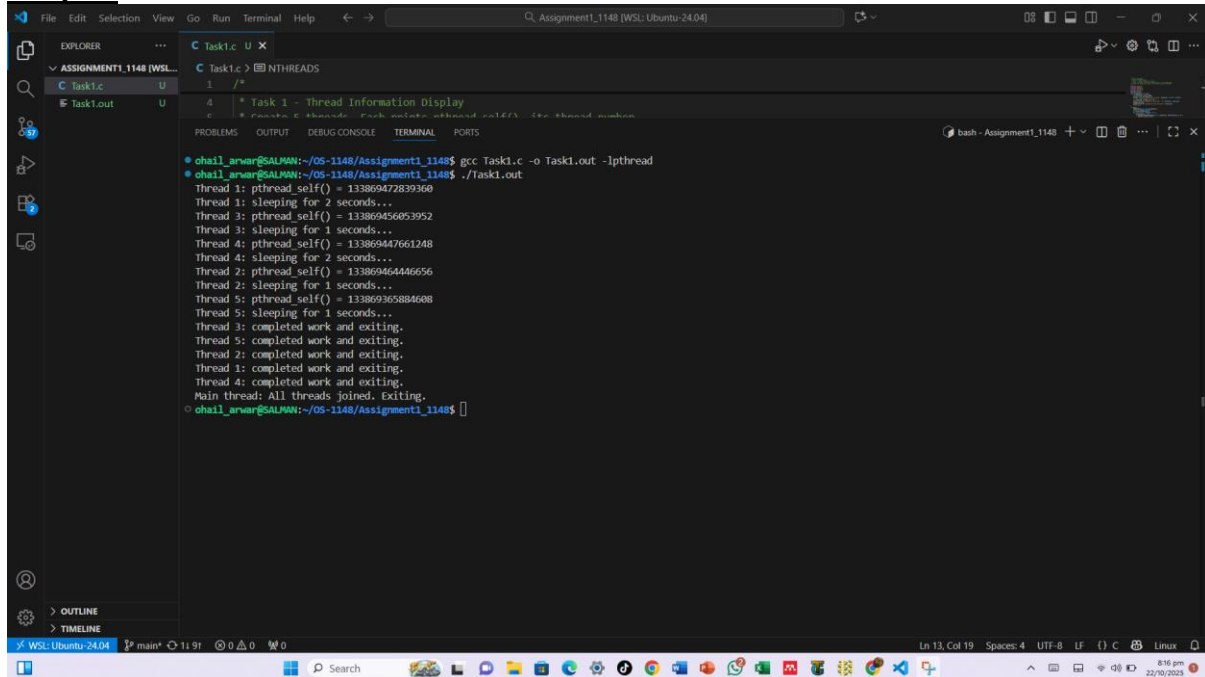
Section A: Programming Tasks

Program 1: Thread Information Display

Code:

```
1  /*
2   * Name: Eman Babar
3   * Reg No: 23-NTU-CS-1148
4   * Task 1 - Thread Information Display
5   * Create 5 threads. Each prints pthread_self(), its thread number,
6   * sleeps for random 1-3 seconds, then prints completion message.
7   */
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <pthread.h>
11 #include <unistd.h>
12 #include <time.h>
13 #define NTHREADS 5
14 void *thread_func(void *arg) {
15     int thread_num = *(int *)arg;
16     pthread_t tid = pthread_self();
17     printf("Thread %d: pthread_self() = %lu\n", thread_num, (unsigned long)tid);
18     /* sleep random between 1-3 seconds */
19     int sleep_time = (rand() % 3) + 1;
20     printf("Thread %d: sleeping for %d seconds...\n", thread_num, sleep_time);
21     sleep(sleep_time);
22     printf("Thread %d: completed work and exiting.\n", thread_num);
23     return NULL;
24 }
25 int main() {
26     srand(time(NULL) ^ (unsigned) getpid());
27     pthread_t threads[NTHREADS];
28     int thread_args[NTHREADS];
29     for (int i = 0; i < NTHREADS; ++i) {
30         thread_args[i] = i + 1;
31         if (pthread_create(&threads[i], NULL, thread_func, &thread_args[i]) != 0) {
32             perror("pthread_create");
33             exit(EXIT_FAILURE);
34         }
35     }
36     /* join all threads */
37     for (int i = 0; i < NTHREADS; ++i) {
38         pthread_join(threads[i], NULL);
39     }
40     printf("Main thread: All threads joined. Exiting.\n");
41     return 0;
42 }
43
```

Output:



```
Task1.c U X
1 /*
4 * Task 1 - Thread Information Display
* Create 5 threads. Each prints thread self's ID, the thread number

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Assignment1_1148

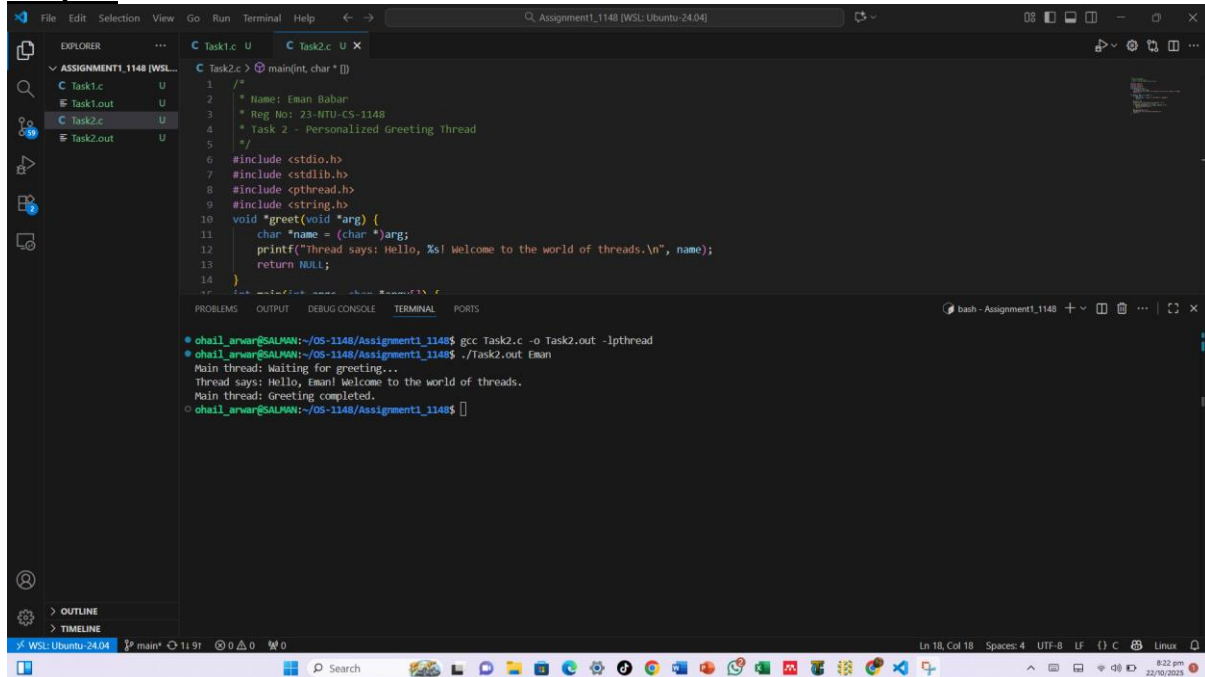
ohail_arwar@SALMAN:~/05-1148/Assignment1_1148$ gcc Task1.c -o Task1.out -lpthread
ohail_arwar@SALMAN:~/05-1148/Assignment1_1148$ ./Task1.out
Thread 1: pthread_self() = 133869472839360
Thread 1: sleeping for 2 seconds...
Thread 3: pthread_self() = 133869456053952
Thread 3: sleeping for 1 seconds...
Thread 4: pthread_self() = 133869447661248
Thread 4: sleeping for 2 seconds...
Thread 2: pthread_self() = 133869464446656
Thread 2: sleeping for 1 seconds...
Thread 5: pthread_self() = 133869365884608
Thread 5: sleeping for 1 seconds...
Thread 3: completed work and exiting.
Thread 5: completed work and exiting.
Thread 2: completed work and exiting.
Thread 1: completed work and exiting.
Thread 4: completed work and exiting.
Main thread: All threads joined. Exiting.
ohail_arwar@SALMAN:~/05-1148/Assignment1_1148$
```

Program 2: Personalized Greeting Thread

Code:

```
1 /*
2  * Name: Eman Babar
3  * Reg No: 23-NTU-CS-1148
4  * Task 2 - Personalized Greeting Thread
5  */
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <pthread.h>
9 #include <string.h>
10 void *greet(void *arg) {
11     char *name = (char *)arg;
12     printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
13     return NULL;
14 }
15 int main(int argc, char *argv[]) {
16     if (argc < 2) {
17         fprintf(stderr, "Usage: %s <YourName>\n", argv[0]);
18         return 1;
19     }
20     pthread_t tid;
21     printf("Main thread: Waiting for greeting...\n");
22     /* pass the provided name to thread */
23     if (pthread_create(&tid, NULL, greet, argv[1]) != 0) {
24         perror("pthread_create");
25         return 1;
26     }
27     pthread_join(tid, NULL);
28     printf("Main thread: Greeting completed.\n");
29     return 0;
30 }
31
```

Output:



The screenshot shows a Visual Studio Code editor window with a C program named Task2.c. The program includes headers for stdio, stdlib, pthread, and string. It defines a greet function that prints a message and returns NULL. The main function calls pthread_create to start a new thread that calls greet. The terminal output shows the compilation and execution of the program, resulting in the message: "Thread says: Hello, Eman! Welcome to the world of threads."

```
1 /*
2  * Name: Eman Babar
3  * Reg No: 23-NTU-CS-1148
4  * Task 2 - Personalized Greeting Thread
5  */
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <pthread.h>
9 #include <string.h>
10 void *greet(void *arg) {
11     char *name = (char *)arg;
12     printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
13     return NULL;
14 }
15
16 int main() {
17     pthread_t tid;
18     int *arg = malloc(sizeof(int));
19     *arg = 1;
20     if (pthread_create(&tid, NULL, greet, arg) != 0) {
21         perror("pthread_create");
22         free(arg);
23         return 1;
24     }
25     pthread_join(tid, NULL);
26     free(arg);
27     printf("Main thread: Work completed.\n");
28     return 0;
29 }
```

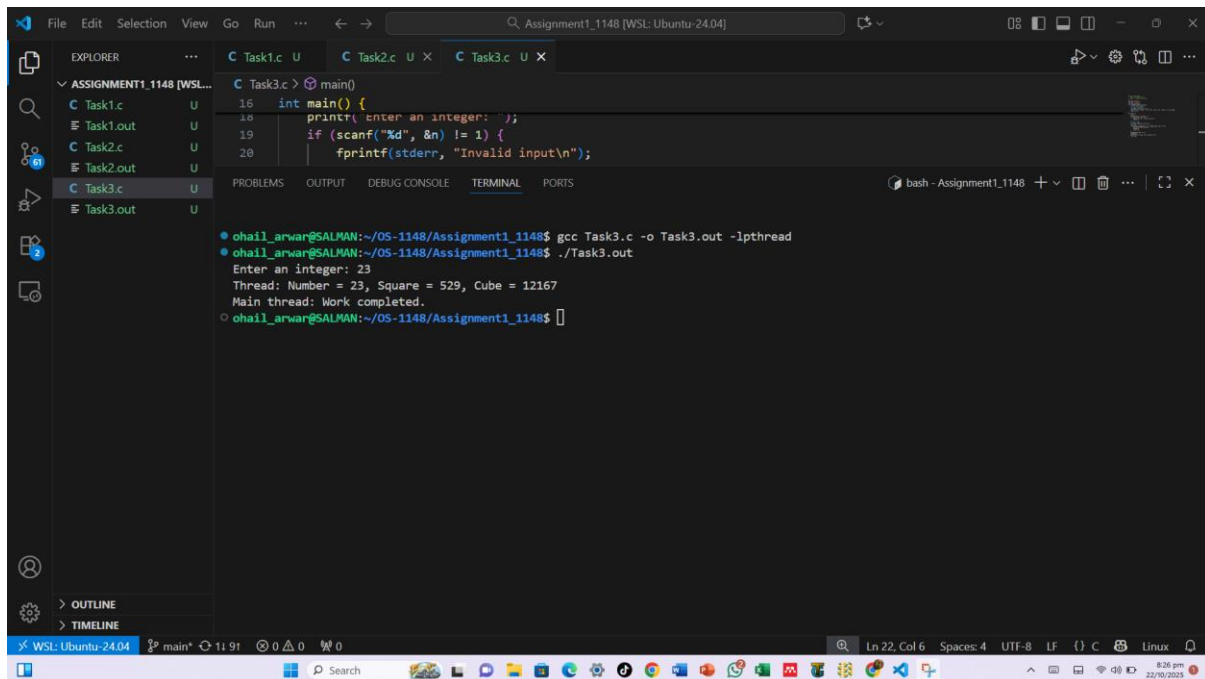
```
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ gcc Task2.c -o Task2.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ ./Task2.out Eman
Main thread: Waiting for greeting...
Thread says: Hello, Eman! Welcome to the world of threads.
Main thread: Greeting completed.
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$
```

Program 3: Number Info Thread

Code:

```
1 /*
2  * Name: Eman Babar
3  * Reg No: 23-NTU-CS-1148
4  * Task 3 - Number Info Thread
5  */
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <pthread.h>
9 void *number_info(void *arg) {
10     int n = *(int *)arg;
11     long sq = (long)n * n;
12     long cube = (long)n * n * n;
13     printf("Thread: Number = %d, Square = %ld, Cube = %ld\n", n, sq, cube);
14     return NULL;
15 }
16 int main() {
17     int n;
18     printf("Enter an integer: ");
19     if (scanf("%d", &n) != 1) {
20         fprintf(stderr, "Invalid input\n");
21         return 1;
22     }
23     pthread_t tid;
24     int *arg = malloc(sizeof(int));
25     *arg = n;
26     if (pthread_create(&tid, NULL, number_info, arg) != 0) {
27         perror("pthread_create");
28         free(arg);
29         return 1;
30     }
31     pthread_join(tid, NULL);
32     free(arg);
33     printf("Main thread: Work completed.\n");
34     return 0;
35 }
36
```

Output:



```
File Edit Selection View Go Run ... Assignment1_1148 [WSL: Ubuntu-24.04]
EXPLORER
  ASSIGNMENT1_1148 [WSL: Ubuntu-24.04]
    Task1.c
    Task1.out
    Task2.c
    Task2.out
    Task3.c
    Task3.out
  C Task3.c U
  C Task2.c U X
  C Task3.c U X
  C Task3.c > main()
    16 int main() {
    18     printf("Enter an integer: ");
    19     if (scanf("%d", &n) != 1) {
    20         fprintf(stderr, "Invalid input\n");
    21     }
    22 }
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  bash - Assignment1_1148
  ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ gcc Task3.c -o Task3.out -lpthread
  ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ ./Task3.out
  Enter an integer: 23
  Thread: Number = 23, Square = 529, Cube = 12167
  Main thread: Work completed.
  ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$
```

Program 4: Thread Return Values

Code:

```
1  /*
2  * Name: Eman Babar
3  * Reg No: 23-NTU-CS-1148
4  * Task 4 - Thread Return Values (Factorial)
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <pthread.h>
9  void *compute_factorial(void *arg) {
10     int n = *(int *)arg;
11     unsigned long long *result = malloc(sizeof(unsigned long long));
12     *result = 1;
13     for (int i = 2; i <= n; ++i) *result *= i;
14     return (void *)result;
15 }
16 int main() {
17     int n;
18     printf("Enter a non-negative integer: ");
19     if (scanf("%d", &n) != 1 || n < 0) {
20         fprintf(stderr, "Invalid input\n");
21         return 1;
22     }
23     pthread_t tid;
24     int *arg = malloc(sizeof(int));
25     *arg = n;
26     if (pthread_create(&tid, NULL, compute_factorial, arg) != 0) {
27         perror("pthread_create");
28         free(arg);
29         return 1;
30     }
31     unsigned long long *res;
32     pthread_join(tid, (void **)&res);
33     printf("Main thread: Factorial of %d = %llu\n", n, *res);
34     free(res);
35     free(arg);
36     return 0;
37 }
```

Output:

```
Task4.c > compute_factorial(void *)
1 /*
5 */

ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ gcc Task4.c -o Task4.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ ./Task4.out
Enter a non-negative integer: 45
Main thread: Factorial of 45 = 9649395409222631424
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ gcc Task4.c -o Task4.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ ./Task4.out
Enter a non-negative integer: 2
Main thread: Factorial of 2 = 2
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$
```

Program 5: Struct-Based Thread Communication

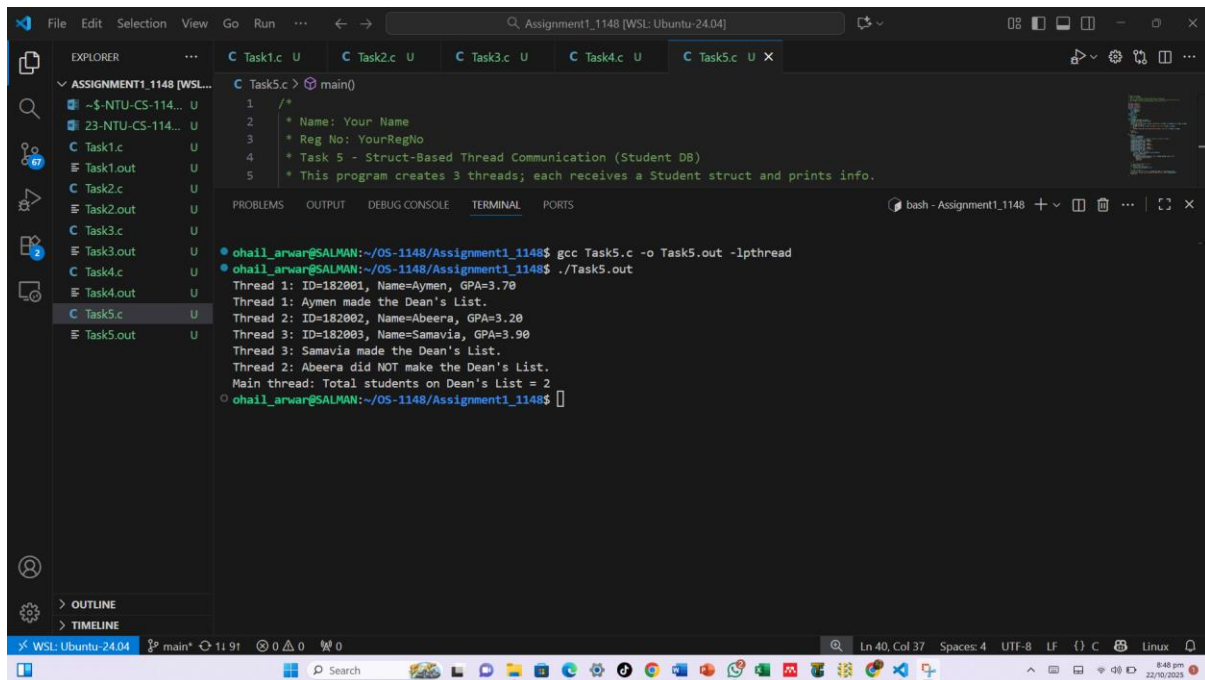
Code:

```

1  /*
2  * Name: Your Name
3  * Reg No: YourRegNo
4  * Task 5 - Struct-Based Thread Communication (Student DB)
5  * This program creates 3 threads; each receives a Student struct and prints info.
6  * Main thread counts how many made Dean's list (GPA >= 3.5).
7  */
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <pthread.h>
12 typedef struct {
13     int student_id;
14     char name[50];
15     float gpa;
16 } Student;
17 typedef struct {
18     Student s;
19     int index;
20 } ThreadArg;
21 void *student_thread(void *arg) {
22     ThreadArg *t = (ThreadArg *)arg;
23     Student st = t->s;
24     printf("Thread %d: ID=%d, Name=%s, GPA=%.2f\n", t->index, st.student_id, st.name, st.gpa);
25     if (st.gpa >= 3.5f) {
26         printf("Thread %d: %s made the Dean's List.\n", t->index, st.name);
27     } else {
28         printf("Thread %d: %s did NOT make the Dean's List.\n", t->index, st.name);
29     }
30     free(t);
31     return NULL;
32 }
33 int main() {
34     pthread_t threads[3];
35     Student students[3];
36     /* Example students -- replace or gather via input as needed */
37     students[0].student_id = 182001;
38     strcpy(students[0].name, "Aymen");
39     students[0].gpa = 3.7f;
40     students[1].student_id = 182002;
41     strcpy(students[1].name, "Abeera");
42     students[1].gpa = 3.2f;
43     students[2].student_id = 182003;
44     strcpy(students[2].name, "Samavia");
45     students[2].gpa = 3.9f;
46     for (int i = 0; i < 3; ++i) {
47         ThreadArg *arg = malloc(sizeof(ThreadArg));
48         arg->s = students[i];
49         arg->index = i + 1;
50         if (pthread_create(&threads[i], NULL, student_thread, arg) != 0) {
51             perror("pthread_create");
52             free(arg);
53             return 1;
54         }
55     }
56     /* join and count Dean's List */
57     int deans_count = 0;
58     for (int i = 0; i < 3; ++i) {
59         pthread_join(threads[i], NULL);
60     }
61     /* simple recount */
62     for (int i = 0; i < 3; ++i) if (students[i].gpa >= 3.5f) deans_count++;
63     printf("Main thread: Total students on Dean's List = %d\n", deans_count);
64     return 0;
65 }

```

Output:



```
1 /*
2  * Name: Your Name
3  * Reg No: YourRegNo
4  * Task 5 - Struct-Based Thread Communication (Student DB)
5  * This program creates 3 threads; each receives a Student struct and prints info.
6  */
7
8 #include <stdio.h>
9 #include <pthread.h>
10
11 struct Student {
12     int ID;
13     char Name[50];
14     float GPA;
15 };
16
17 pthread_t t1, t2, t3;
18 struct Student s1, s2, s3;
19 int count = 0;
20
21 void *thread_func(void *arg) {
22     struct Student *s = (struct Student *)arg;
23     printf("Thread %d: ID=%d, Name=%s, GPA=%f\n",
24            (int)arg, s->ID, s->Name, s->GPA);
25     if (s->GPA >= 3.70) {
26         count++;
27         printf("%s made the Dean's List.\n", s->Name);
28     }
29     return NULL;
30 }
31
32 int main() {
33     s1.ID = 182001; strcpy(s1.Name, "Aymen"); s1.GPA = 3.70;
34     s2.ID = 182002; strcpy(s2.Name, "Abeera"); s2.GPA = 3.20;
35     s3.ID = 182003; strcpy(s3.Name, "Samavia"); s3.GPA = 3.90;
36
37     pthread_create(&t1, NULL, thread_func, (void *)s1.ID);
38     pthread_create(&t2, NULL, thread_func, (void *)s2.ID);
39     pthread_create(&t3, NULL, thread_func, (void *)s3.ID);
40
41     pthread_join(t1, NULL);
42     pthread_join(t2, NULL);
43     pthread_join(t3, NULL);
44
45     printf("Main thread: Total students on Dean's List = %d\n", count);
46     return 0;
47 }
```

```
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ gcc Task5.c -o Task5.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$ ./Task5.out
Thread 1: ID=182001, Name=Aymen, GPA=3.70
Thread 1: Aymen made the Dean's List.
Thread 2: ID=182002, Name=Abeera, GPA=3.20
Thread 3: ID=182003, Name=Samavia, GPA=3.90
Thread 3: Samavia made the Dean's List.
Thread 2: Abeera did NOT make the Dean's List.
Main thread: Total students on Dean's List = 2
ohail_arwar@SALMAN:~/OS-1148/Assignment1_1148$
```

Section B: Short Questions

Question 1:

Define an Operating System in a single line.

Solution:

A fundamental software that acts as an intermediary between a computer's hardware and the user, managing all hardware resources and provides services to programs and users.

Example:

Desktop: Microsoft Windows, macOS and Linux.

Mobile: Android, iOS.

Question 2:

What is the primary function of the CPU scheduler?

Solution:

The primary function of the CPU scheduler is to select a process from the ready queue and allocate the Central Processing Unit (CPU) to it. This process is vital for multitasking operating systems, which must juggle multiple programs vying for a single CPU.

Question 3:

List any three states of a process.

Solution:

The three most common states of a process are:

Ready:

A process in the ready state has been loaded into main memory and is ready to be executed.

Running:

A process in the running state is currently being executed by the CPU.

Blocked:

A process enters the blocked state when it must wait for some event or resource before it can continue.

Question 4:

What is meant by a Process Control Block (PCB)?

Solution:

A Process Control Block (PCB) is a data structure used by an operating system to manage a

process. It holds all the information necessary to control and execute a process, including its state, program counter, CPU registers, memory management details and I/O status. The OS creates a unique PCB for each process and uses it to store context when switching between them.

Question 5:

Differentiate between a process and a program.

Solution:

Feature	Program	Process
Definition	A program is a set of instructions written in a programming language that performs a specific task.	A process is an instance of a program that is being executed by the computer.
State	It is just code stored on disk.	It has a state, is running and uses CPU, memory and other resources.
Storage	It is stored in secondary memory like a hard disk.	It exists in main memory while executing.
Multiplicity	One program can be executed multiple times, creating multiple processes.	Each process represents one running instance of a program.
Example	Chrome.exe stored on disk.	When we open chrome, each browser window/tab is a process of chrome.exe.

Question 6:

What do you understand by context switching?

Solution:

Context switching is the process of a CPU or a person shifting focus from one task to another. It allows an operating system to give the appearance of multiple processes running at once, even though the CPU is only executing one at any given moment.

Question 7:

Define CPU utilization and throughput.

Solution:

CPU Utilization:

The amount of time the CPU spends actively working on processes versus being idle. It is measured as percentages.

Example:

If the CPU is running a single, long-running calculation, it might have 100% utilization.

Throughput:

The number of processes that are successfully completed per unit of time. It is measured in “tasks per second” or a similar unit.

Example:

A system that can complete 500 transactions per second has a throughput of 500 transactions/second.

Question 8:

What is the turnaround time of a process?

Solution:

Turnaround time of a process is the total time it takes from its submission to its completion. This includes the time a process spends waiting, the time it takes to be executed, and any other delays.

Calculation Formula:

It is calculated using following formula:

Turnaround time = Completion time – Arrival time

Question 9:

How is waiting time calculated in process scheduling?

Solution:

Sum of time a process spends waiting in the ready queue is termed as waiting time.

Formula:

It is calculated by using following formula:

Waiting time = Turnaround time – Service Time

Question 10:

Define response time in CPU scheduling.

Solution:

Response time is the duration from when a process arrives in the ready queue until it first gets the CPU. It is a measure of a system's responsiveness, indicating how quickly a user receives an initial reaction from the system after submitting a request.

Formula:

Response time = Time the process gets CPU – Arrival time

Question 11:

What is preemptive scheduling?

Solution:

Preemptive scheduling is a process where a running task can be interrupted by another, higher-priority task, even if the first task is not finished. This allows the CPU to be given to a more critical process, such as one that has just arrived with a higher priority or a shorter execution time by moving the currently executing process to the ready queue. It is used in modern operating systems like Windows, Linux and macOS to ensure responsiveness.

Question 12:

What is non-preemptive scheduling?

Solution:

It is a type of CPU scheduling in an operating system where a process runs until it completes its CPU burst or voluntarily switches to a waiting state. Once a process is allocated the CPU, it cannot be interrupted by other processes, even if a higher-priority arrives, making it inflexible but simpler to implement.

Question 13:

State any two advantages of the Round Robin scheduling algorithm.

Solution:

Following are the two advantages of Round Robin scheduling algorithm:

➤ **Fairness:**

Every process gets an equal share of CPU time in a cyclic order, so no process is starved of CPU.

➤ **Good response time for time-sharing systems:**

Since each process gets the CPU for small time quantum, interactive users experience quick responses, making Round Robin suitable for multitasking and time-sharing environments.

Question 14:

Mention one major drawback of the Shortest Job First (SJF) algorithm.

Solution:

A major drawback of Shortest Job First (SJF) algorithm is that it can cause starvation, where long processes may never get to execute if there is a continuous stream of shorter processes arriving. Furthermore, it is difficult to implement in practice because the exact burst time of a process can't be known in advance.

Question 15:

Define CPU idle time.

Solution:

CPU idle time is the period during which a processor is available but not actively executing tasks, meaning it is not performing any computations. This unused capacity is often measured by the OS, which may run a low-priority “idle task” to track the time and allow the CPU to enter low-power states save energy.

Measurement:

In UNIX/LINUX systems, it is often shown in the **top** command as the **id** field.

Question 16:

State two common goals of CPU scheduling algorithms.

Solution:

Two common goals of CPU scheduling algorithms are:

➤ **Maximize CPU utilization:**

The goal is to keep CPU as busy as possible to ensure that is not idle. It can be achieved by switching to another process when the current one is waiting for something, like I/O operation.

➤ **Minimize Response Time:**

This is the time from when a request is submitted until the first response is produced. Algorithms that minimize response time are crucial for interactive systems where users expect immediate feedback.

Question 17:

List two possible reasons for process termination.

Solution:

The two possible reasons are as follows:

➤ **Normal completion:**

The process finishes execution successfully.

➤ **Error or exception:**

The process is terminated due to an error.

Question 18:

Explain the purpose of the wait () and exit () system calls.

Solution:

➤ **wait ():** used by a parent process to wait until one child processes finishes execution, allowing the parent to collect the child’s exit status.

➤ **exit ():** used by a parent process to terminate its execution and release resources.

Question 19:

Differentiate between shared memory and message-passing models of inter-process communication.

Solution:

Shared memory	Message passing
Processes communicate by sharing a common memory region.	Processes communicate by sending and receiving messages.
Faster communication once memory is shared.	Slower due to message copying and system calls.
Requires synchronization.	Synchronization is implicit in message exchange.

Question 20:

Differentiate between a thread and a process.

Solution:

Process	Thread
Independent execution unit with its own memory space.	A smaller unit of a process that shares memory with other threads.
Heavyweight, more overhead.	Lightweight, less overhead.
Inter-process communication is slower.	Inter-thread communication is faster.

Question 21:

Define multithreading.

Solution:

Multithreading is the ability of a CPU or a single process to execute multiple threads concurrently, improving performance and resource utilization.

By breaking a program into smaller independent threads, it can perform several tasks at once, such as web browser displaying a webpage while a separate thread loads an image or a document editor saving in the background while you continue typing.

Question 22:

Explain the difference between a CPU-bound process and an I/O-bound process.

Solution:

CPU-bound	I/O-Bound
Spends most of its time performing computations.	Spends most of its time waiting I/O operations.
High CPU utilization.	Low CPU utilization.

Question 23:

What are the main responsibilities of the dispatcher?

Solution:

Following are the three main responsibilities of a dispatcher:

- It switches the CPU from one process to another.
- It saves and restores process states during a context switch.
- Transfers control to the new process.

Question 24:

Define starvation and aging in process scheduling.

Solution:

Starvation: A process waits indefinitely because it never gets CPU time.

Aging: A technique to prevent starvation by gradually increasing the priority of waiting processes.

Question 25:

What is a time quantum (or time slice)?

Solution:

A quantum time is the fixed amount of CPU time allotted to each process in Round Robin scheduling before it is pre-empted.

Question 26:

What happens when the time quantum is too large or too small?

Solution:

Top large: System behaves like First-Come, First-Served (poor response time).

Too small: Too many context switches (high overhead).

Question 27:

Define the turnaround ratio (TR/TS).

Solution:

$$\text{Turnaround Ratio} = \frac{\text{Turnaround Time (TR)}}{\text{Service Time (TS)}}$$

It shows how efficiently a process completes compared to its actual CPU service time.

Question 28:

What is the purpose of a ready queue?

Solution:

The ready queue holds all processes that are in main memory and ready to execute but are currently waiting for CPU allocation.

Question 29:

Differentiate between a CPU burst and an I/O burst.

Solution:

Aspect	CPU Burst	I/O Burst
Definition	A period during which a process is using the CPU to execute instructions.	A period during which a process is waiting for I/O operations to complete.
Resource used	CPU	I/O devices
Speed	Usually very fast	Generally slower compared to CPU bursts.
Example	Calculating an expression, sorting data	Reading a file from disk, waiting for user input.

Question 30:

Differentiate between a CPU burst and an I/O burst.

Solution:

Feature	CPU Burst	I/O Burst
Definition	Time during which a process is executing instructions on the CPU.	Time during which a process is waiting for input/output operations to complete.
Resource Used	Uses CPU intensively.	Uses I/O devices like disk, printer and keyboard.
Speed	Very fast since CPU operates at high speed.	Slower, because I/O devices are slower as compared to CPU.
Example	Performing arithmetic operations in a program.	Reading a file or waiting for user input.

Question 31:

Which scheduling algorithm is starvation-free, and why?

Solution:

Round-Robin scheduling algorithm is starvation free because every process gets a fixed time slice cyclically, ensuring no process is indefinitely delayed.

Question 32:

Outline the main steps involved in process creation in UNIX.

Solution:

Following are the main steps of creation of UNIX:

- **fork ():**
creates a new process child identical to the parent.
- **exec ():**
replaces the child's memory with the new program.
- **wait ():**
parent waits for child to finish.
- **exit ():**
child terminates and returns status to parent.

Question 33:

Define zombie and orphan processes.

Solution:

Zombie process:

A process that has finished its execution and exited, but its entry remains in the process table because the parent has not yet acknowledged its termination.

Example:

A parent process creates a child's process, and the child exits immediately. The parent, however, continues running and doesn't check for child's status. The child becomes a zombie until the parent eventually call wait () to clear its entry.

Orphan process:

It is a child process whose parent process terminates before it does; the orphan process is then adopted by the init process (PID 1).

Example:

A parent process starts a child process but then exits prematurely. The OS detects that the parent is gone and transfers the child to the "init" process (PID 1, which will eventually clean it up once it's done.

Question 34:

Differentiate between Priority Scheduling and Shortest Job First (SJF).

Solution:

Priority Scheduling	Shortest Job First (SJF)
CPU is assigned based on priority level.	CPU is assigned based on shortest burst time.
Can cause starvation of low-priority process.	May cause starvation of long processes.

Question 35:

Define context switch time and explain why it is considered overhead.

Solution:

Context switch time is the time taken by the CPU to save the state of the current process and load the state of the next process.

It is considered overhead because no useful work is done during this time.

Question 36:

List and briefly describe the three levels of schedulers in an Operating System.

Solution:

- **Long-term Scheduler:** decides which jobs to admit into ready queue.
- **Short-term Scheduler:** selects which process gets the CPU next.
- **Medium-term Scheduler:** handles suspension and resumption of processes (swapping).

Question 37:

Differentiate between User Mode and Kernel Mode in an Operating System.

Solution:

Feature	User mode	Kernel mode
Definition	Mode in which user applications run.	Mode in which operating system kernel runs.
Access to hardware	Has limited access to hardware and system resources.	Has full access to all hardware and system resources.
Program type	Used by user programs like browsers, word processors, etc.	Used by system programs like device drivers, memory manager, scheduler, etc.
Mode switching	Switches to kernel mode when a system call or	Switches to user mode when control is returned to user

	interrupts occurs.	program.
Example:	Running MS Word, Chrome or a game.	Executing file management, memory allocation, or I/O control.

Section C: Technical/Analytical Questions

Question 1:

Describe the complete life cycle of a process with a neat diagram showing transitions between New, Ready, Running, Waiting, and Terminated states in a single line.

Solution:

A process passes through several states during its execution, from creation to termination.

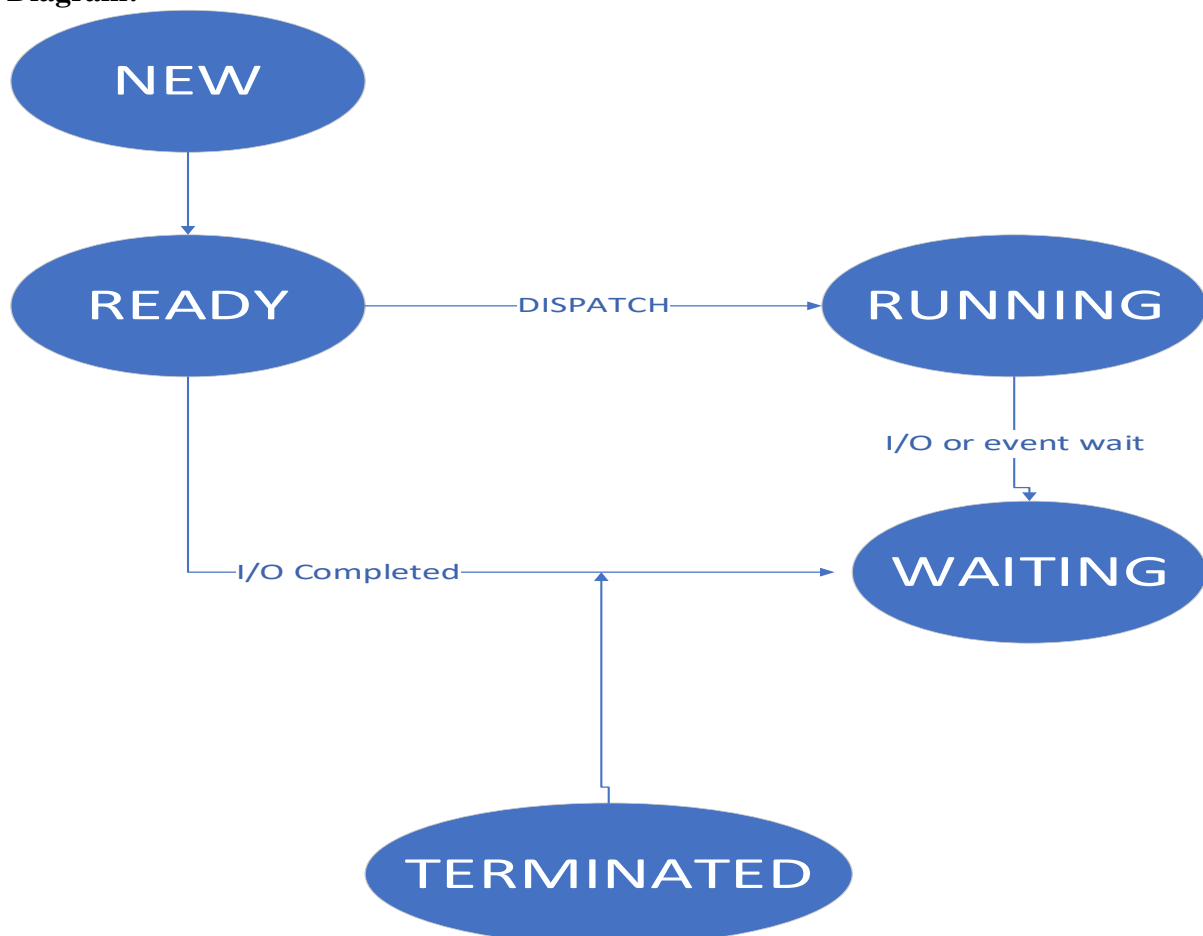
States & Transitions:

- **New:** Process is being created.
- **Ready:** Process is loaded into memory and waiting for CPU allocation.
- **Running:** Process is currently being executed by the CPU.
- **Waiting (or blocked):** Process is waiting for some event (like I/O completion).
- **Terminated:** Process has finished execution.

Transitions:

- **New → Ready:** Process is admitted by the long-term scheduler.
- **Ready → Running:** CPU scheduler dispatches the process.
- **Running → Waiting:** Process requests I/O or waits for an event.
- **Waiting → Ready:** Event is completed; process moves back to ready queue.
- **Running → Terminated:** Process execution completed.

Diagram:



Question 2:

Write a short note on context switch overhead and describe what information must be saved and restored.

Solution:

A context switch occurs when the CPU switches from executing one process to another.

Overhead:

During a context switch, the CPU must save the state of the currently running process and load the state of the next process.

This consumes CPU time and no useful work is done during this period. Hence, called context switch overhead.

Information Saved & Restored:

- Program counter
- CPU registers
- Stack pointer
- Process state
- Memory management information (e.g., page tables).
- Accounting information (e.g., CPU time used).

In short, Context switching allows multitasking but increases overhead due to saving and restoring process states.

Question 3:

List and explain the components of a Process Control Block (PCB).

Solution:

A process control block (PCB) is a data structure maintained by the OS to store information about a process.

Components:

Components	Description
Process ID (PID)	Unique identifier for each process.
Process State	Indicates if process is New, Ready, Running, Waiting or Terminated.
Program Counter	Holds address of next instruction to execute.
CPU Registers	Stores process's register values.
Memory Management information	Includes base/limit registers, page tables etc.
Accounting Information	CPU usage, process priority, execution time, etc.
I/O Status information	List of I/O devices allocated or files opened.

In short, PCB acts as a record card for each process, enabling suspension and resumption correctly.

Question 4:

Differentiate between Long-Term, Medium-Term, and Short-Term Schedulers with examples.

Solution:

Aspect	Long-Term	Short-Term	Medium-Term
Function	Selects processes from secondary storage and loads them into memory.	Selects one of the ready processes for CPU execution.	Temporarily removes and later reintroduces processes to control load.
Type/Name	Job Scheduler	CPU scheduler	Process Swapping Scheduler.

Speed	Slowest of all	Fastest; runs most frequently.	Intermediate speed between long and short term.
Multiprogramming Control	Controls the degree of multiprogramming.	Gives less control over the degree of multiprogramming.	Reduces the degree of multiprogramming when memory is overloaded.
Key Operation	Admits new job into memory.	Dispatches ready processes to CPU.	Suspends and later resumes processes.
Example	Loading jobs from disk to main memory.	Time-sharing CPU allocation.	Swapping out a suspended process.

Question 5:

Explain CPU Scheduling Criteria (Utilization, Throughput, Turnaround, Waiting, and Response) and their optimization goals.

Solution:

Criterion	Meaning	Goal (optimization)
CPU Utilization	Percentage of time CPU is busy.	Maximize utilization keep CPU as busy as possible.
Throughput	Number of processes completed per unit time.	Maximize throughput.
Turnaround Time	Total time taken from process submission to completion.	Minimize turnaround time.
Waiting Time	Total time a process spends waiting in the ready queue.	Minimize waiting time.
Response Time	Time between submission and first response/output.	Minimize response time

In short, good CPU scheduling tries to maximize CPU utilization and throughput, while minimizing waiting, turnaround and response times.

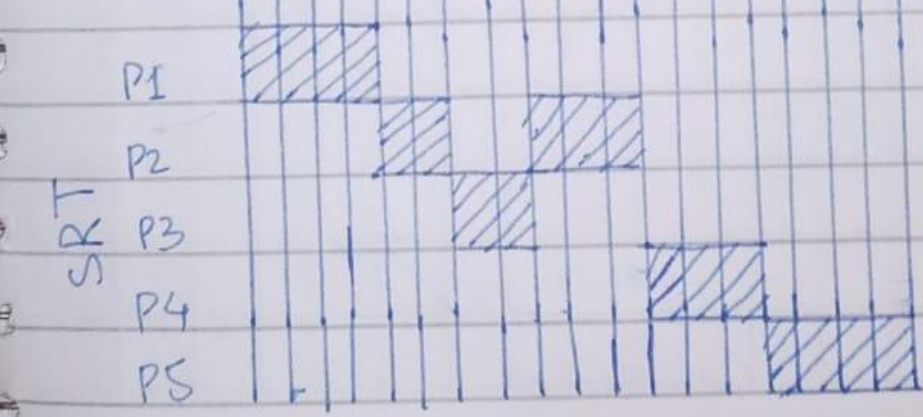
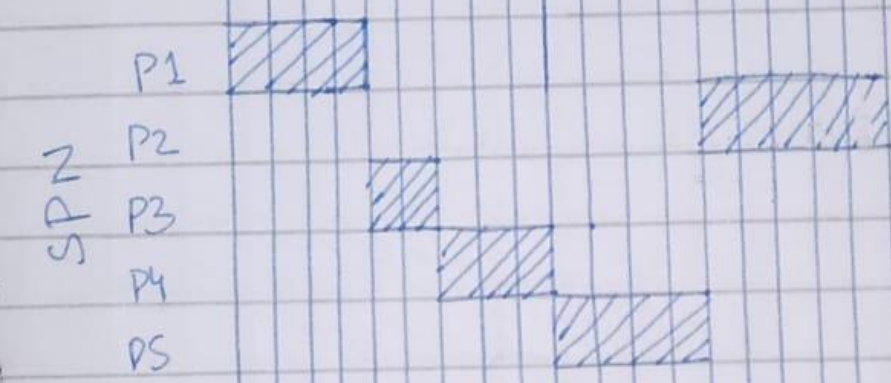
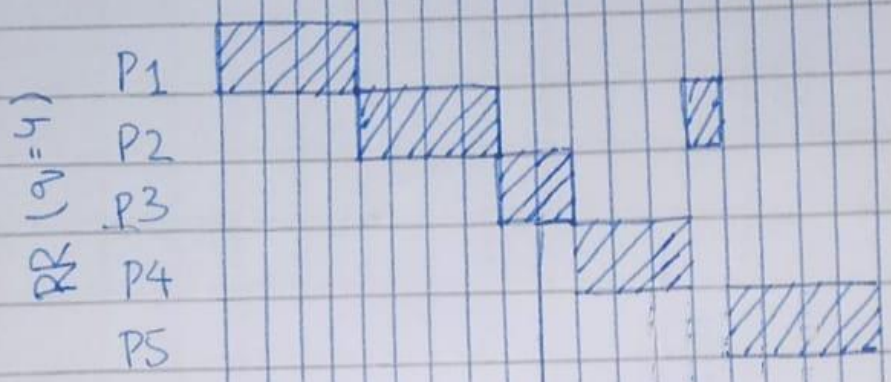
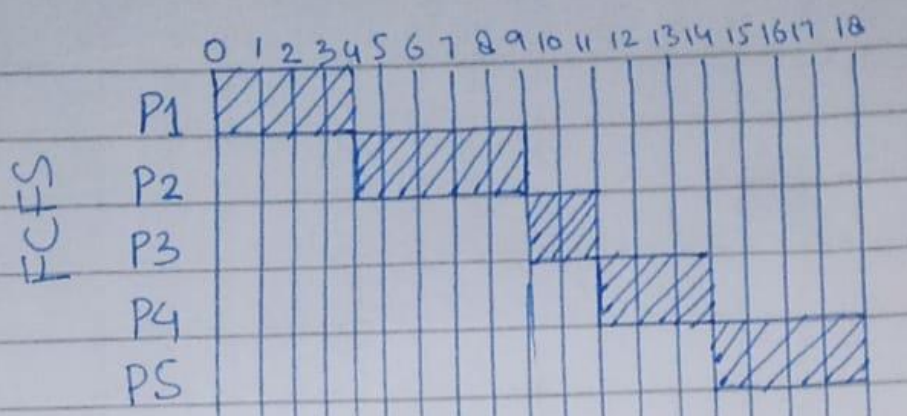
Section D: CPU Scheduling Calculations

Perform the following calculations for each part (A–C).

- Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.
- Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.
- Compare average values and identify which algorithm performs best.

PART A:

Process	Arrival Time	Service Time
P1	0	4
P2	2	5
P3	4	2
P4	6	3
P5	9	4



FCFS

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	2	5	5	5
Turnaround Time	4	7	7	8	9
TR/TS	1	1.4	3.5	2.67	2.25

CPU Idle Time: 0

c) Average Values:

Average Waiting Time: 3.4

Average Turnaround Time: 7

Average TR/TS: 2.164

Round Robin (RR): q = 4

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	7	4	4	5
Turnaround Time	4	12	6	7	9
TR/TS	1	2.4	3	2.33	2.25

c) Average Values:

Average Waiting Time: 4

Average Turnaround Time: 7.6

Average TR/TS: 2.196

Shortest Job First (SJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	11	0	0	0
Turnaround Time	4	16	2	3	4
TR/TS	1	3.2	1	1	1

c) Average Values:

Average Waiting Time: 2.2

Average Turnaround Time: 7.6

Average TR/TS: 1.44

Shortest Remaining Job First (SRJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	4	2	5	5
--------------	---	---	---	---	---

Turnaround Time	4	9	4	8	9
TR/TS	1	1.8	2	2.7	2.3

c) Average Values:

Average Waiting Time: 3.2

Average Turnaround Time: 7.6

Average TR/TS: 1.96

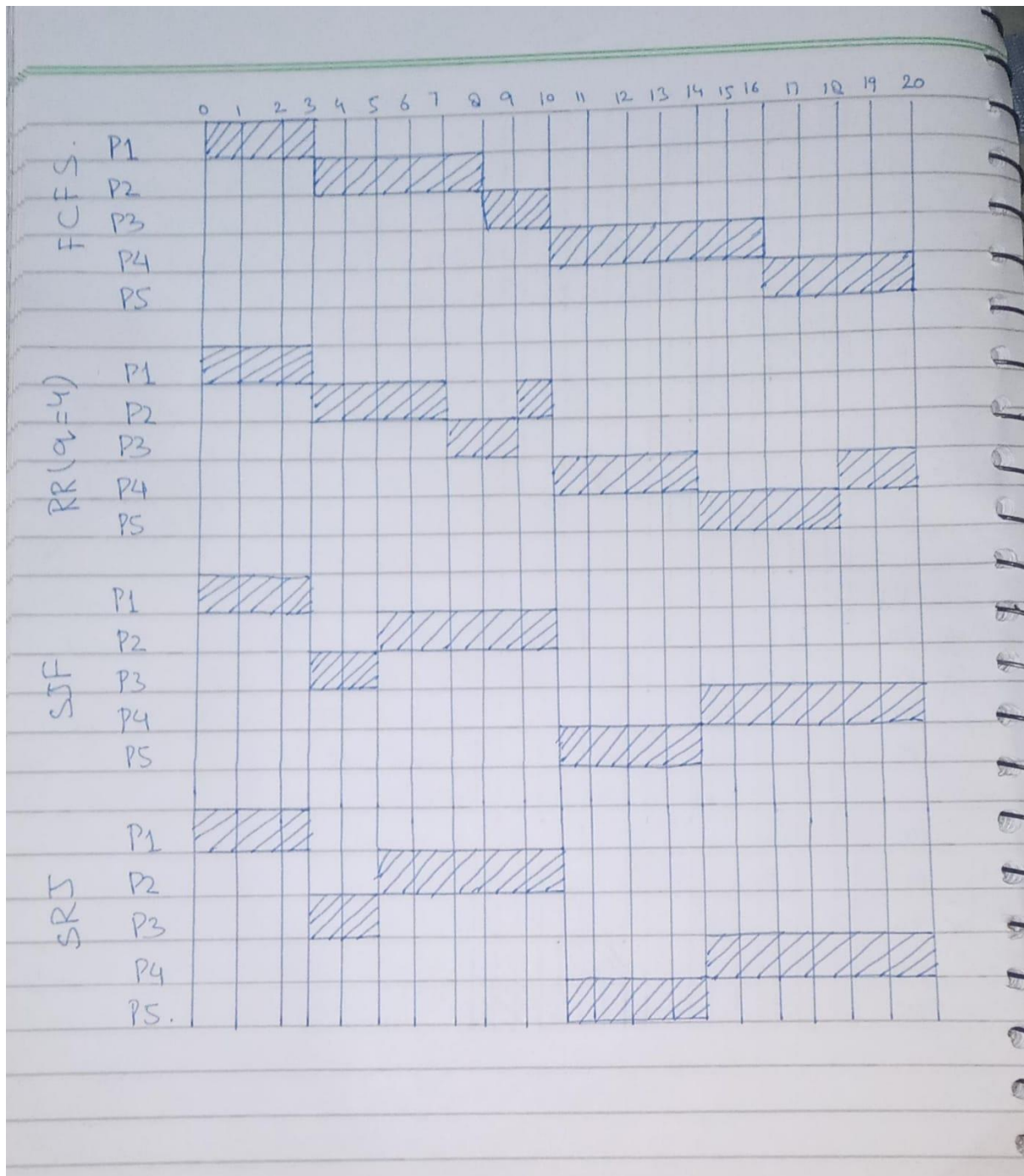
Best Performance:

SJF is best algorithm because it allocates processes having shortest jobs.

PART B:

Process	Arrival Time	Service Time
P1	0	3
P2	1	5
P3	3	2
P4	9	6
P5	10	4

CPU idle time = 0



FCFS

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	2	5	1	6
Turnaround Time	3	7	7	7	10
TR/TS	1	1.4	3.5	1.2	2.5

c) Average Values:

Average Waiting Time: 2.8

Average Turnaround Time: 6.8

Average TR/TS: 1.92

Round Robin (RR): q = 4

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	4	4	5	4
Turnaround Time	3	9	6	11	8
TR/TS	1	1.8	3	1.83	2

c) Average Values:

Average Waiting Time: 3.4

Average Turnaround Time: 7.4

Average TR/TS: 1.93

Shortest Job First (SJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	4	0	5	0
Turnaround Time	3	9	2	11	4
TR/TS	1	1.8	1	1.83	1

c) Average Values:

Average Waiting Time: 1.8

Average Turnaround Time: 5.8

Average TR/TS: 1.33

Shortest Remaining Job First (SRJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	4	0	5	0
Turnaround Time	3	9	2	11	4
TR/TS	1	1.8	1	1.83	1

c) Average Values:

Average Waiting Time: 1.8

Average Turnaround Time: 5.8

Average TR/TS: 1.33

Best Performance:

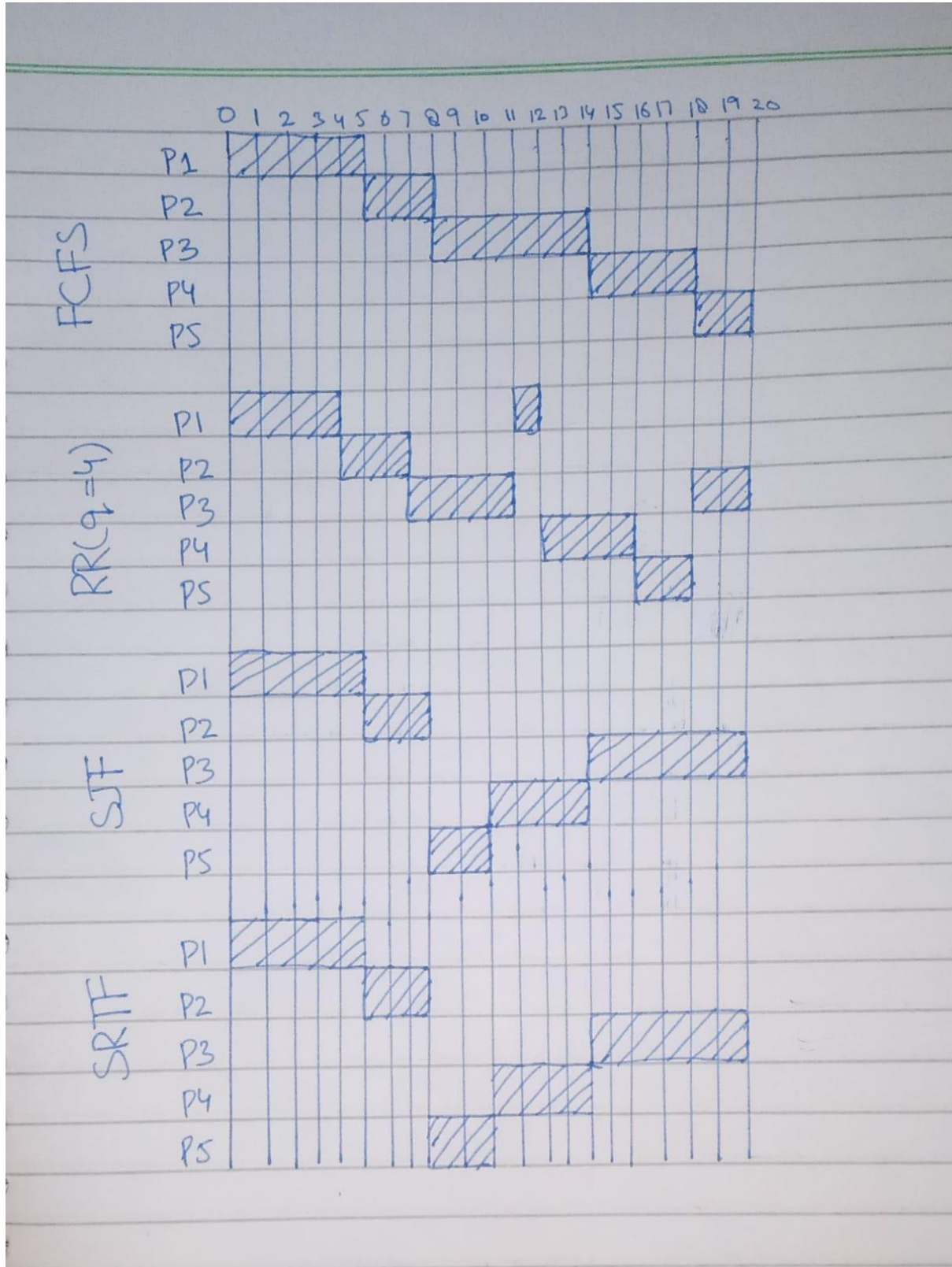
SJF and SRJF are best because they give lowest average turnaround and waiting times.

PART C:

Process	Arrival Time	Service Time
---------	--------------	--------------

P1	0	5
P2	2	3
P3	4	6
P4	6	4
P5	8	2

CPU idle time = 0



FCFS

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	3	4	8	10
Turnaround Time	5	6	10	12	12
TR/TS	1	2	1.667	3	6

c) Average Values:

Average Waiting Time: 5

Average Turnaround Time: 9

Average TR/TS: 2.73

Round Robin (RR): q = 4

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	7	2	10	6	8
Turnaround Time	12	5	16	10	10
TR/TS	2.4	1.667	2.667	2.5	5

c) Average Values:

Average Waiting Time: 6.6

Average Turnaround Time: 10.6

Average TR/TS: 2.84

Shortest Job First (SJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	3	10	4	0
Turnaround Time	5	6	16	8	2
TR/TS	1	2	2.667	2	1

c) Average Values:

Average Waiting Time: 3.4

Average Turnaround Time: 7.4

Average TR/TS: 1.73

Shortest Remaining Job First (SRJF)

b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

Waiting Time = Turnaround Time – Service Time

Turnaround Time = Finish Time – Arrival Time

TR/TS = Turnaround Time/Service Time

Waiting Time	0	3	10	4	0
Turnaround Time	5	6	16	8	2

Time					
TR/TS	1	2	2.667	2	1

c) Average Values:

Average Waiting Time: 3.4

Average Turnaround Time: 7.4

Average TR/TS: 1.73

Best Performance:

SRF and SRJF are the best because they give lowest average waiting time and turnaround time.